Question 1 [5 marks] Choose the best answer among the followings

1. **Which method is practical to perform a single search in an unsorted list of element? elements**
   a. Brute force string matching
   b. Bubble sort
   c. Sequential search
   d. None of the above
2. **Merge sort uses**
   a. Backtracking
   b. Decrease-and-conquer
   c. Divide-and-conquer
   d. Transform and conquer
3. **How many comparisons will be made to sort the array {3,8,5,4,2,10,6} using counting sort?**
   a. 5
   b. 7
   c. 9
   d. 0
4. **In ____, larger solutions for problems are found based upon the solution of a number of smaller problems.**
   a. Divide and conquer
   b. Decrease and conquer
   c. Transform and conquer
   d. All of the above mentioned.
5. **____ is not a balanced search tree**
   a. AVL tree
   b. Red-black tree
   c. Binary tree
   d. None of the above

Question 2 [5 marks]

The following is an insertion search algorithm (in Python)

```
# Insertion sort in Python

def insertionSort(array):

    for step in range(1, len(array)):
        key = array[step]
        j = step - 1


        while j >= 0 and key < array[j]:
            array[j + 1] = array[j]
            j = j - 1

            array[j + 1] = key


data = [11, 6, 1, 5, 3]
insertionSort(data)
print('Sorted Array in Ascending Order:')
print(data)
```

a. . Give a pseudocode representation of this algorithm

```
for step' to n-1 do
    key <- array[step]
     j <- step-1
    while j≥0 and key<array[j] do
        array[j+1] <- array[j]
        j <- j-1
        array[j+1] <- key
    end while
```

b. Trace (by hand) each step of the algorithm until you get a sorted array: **1, 3, 5, 6, 11**

Short form

[11,6,1,5,3]
[6,11,1,5,3]
[1,6,11,5,3]
[1,5,6,11,3]
[1,3,5,6,11]

full form

[11,6,1,5,3]
[6,11,1,5,3]
[6,1,11,5,3]
[1,6,11,5,3]
[1,6,5,11,3]
[1,5,6,11,3]
[1,5,6,3,11]
[1,5,3,6,11]
[1,3,5,6,11]

Question 3 [5 marks]

Given the following algorithm

ALGORITHM *ComparisonCountingSort*($A[0..n-1]$)
//Sorts an array by comparison counting
//Input: Array $A[0..n-1]$ of orderable values
//Output: Array $S[0..n-1]$ of A's elements sorted
// in nondecreasing order
for $i \leftarrow 0$ **to** $n-1$ **do**
    $Count[i] \leftarrow 0$
for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[i] < A[j]$
            $Count[j] \leftarrow Count[j]+1$
        else $Count[i] \leftarrow Count[i]+1$
for $i \leftarrow 0$ **to** $n-1$ **do**
    $S[Count[i]] \leftarrow A[i]$
return S

1. Apply this algorithm to sort the list of numbers: 3 , 2, 1, 4, 6, 7

initially count []

i = 0	0	0	0	0	0	0
i = 1	2	0	0	1	1	1
i = 2		1	0	2	2	2
i = 3			0	3	3	3
i = 4				3	4	4
					4	5
final count[]	2	1	0	3	4	5

So S[] will be [1, 2, 3, 4, 6, 7]

2. Is this algorithm in-place? Demonstrate

No, it uses two extra arrays

Question 4 [5 marks]

a) Write pseudocode for a divide-and-conquer algorithm for finding the position of the largest element in an array of n numbers.

```
FindLargest ( A[0..n-1])
// input: array  A[0..n-1]
// output: largest element  m
m ← A[0]
for i in to  n-1 do
    if  A[i] > m
        m ← A[i]
return m
```

-1

b) Set up and solve a recurrence relation for the number of key comparisons made by your algorithm.

$$\sum_{i=1}^{n-1} 1 = n-1-1+1 = n-1 \in O(n)$$

So we have $n-1$ comparisons

Question 5 [2 marks]

a) For (n Log n)² and log(log n³) pairs of functions, find whether the first function has a lower, same, or higher order of growth (to within a constant multiple) than the second function.

$(n \log n)^2 = n \log n \cdot n \log n$

$\log n^2 = 2 \log n$

So $(n \log n)^2$ has higher order of growth ✓

b) Compare the orders of growth of

$\frac{1}{2}n(n-1)$ and $n^2 \log n$.

$$\lim_{n\to\infty} \frac{\frac{1}{2}n(n-1)}{n^2 \log n} = \frac{1}{2} \lim_{n\to\infty} \frac{n^2\left(1-\frac{1}{n}\right)}{n^2 \log n} = \frac{1}{2} \lim_{n\to\infty} \frac{1}{\log n} = 0$$

So $n^2 \log n$ has higher order of growth ✓

Question 6 [3 marks]

Given the algorithm

```
l ← 0;  r ← n − 1
while l ≤ r do
    m ← ⌊(l + r)/2⌋
    if K = A[m] return m
    else if K < A[m] r ← m − 1
    else l ← m + 1
return −1
```

1. What is its basic operation?

   elements comparision ✓

2. How many times is the basic operation executed?

   $\log n + 1$ ✓

3. What is the efficiency class of this algorithm

   $O(\log n)$ ✓

Question 7 [3 marks]

Given the following two algorithms

```
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 0 to n − 2 do
    min ← i
    for j ← i + 1 to n − 1 do
        if A[j] < A[min]  min ← j
    swap A[i] and A[min]
```

Algorithm A

```
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 0 to n − 2 do
    for j ← 0 to n − 2 − i do
        if A[j + 1] < A[j]  swap A[j] and A[j + 1]
```

Algorithm B

1. What are these algorithms named?
   A is Selection Sort
   B is Bubble Sort
2. Compare their time and space efficiencies.

They both has time of n^2 and space of 1

Question 8 [2 marks]

The following is an insertion search algorithm (in Python)

```
# Insertion sort in Python


def insertionSort(array):

    for step in range(1, len(array)):
        key = array[step]
        j = step - 1


        while j >= 0 and key < array[j]:
            array[j + 1] = array[j]
            j = j - 1

            array[j + 1] = key


data = [4, 5, 2, 1, 11]
insertionSort(data)
print('Sorted Array in Ascending Order:')
print(data)
```

Trace (by hand) each step of the algorithm until you get a sorted array: 1, 2, 4, 5, 11

[4,5,2,1,11]
[2,4,5,1,11]
[1,2,4,5,11]

Short form

[4,5,2,1,11]
[4,2,5,1,11]
[2,4,5,1,11]
[2,4,1,5,11]
[2,1,4,5,11]
[1,2,4,5,11]

full form